CS3025 Compiladores

Laboratorio 9.1 – 9 octubre 2023

Analisis Semantico – Type Checking II

El folder lab9.1 la implementación del parser, printer e interpretre del lenguaje IMP definido por la siguiente sintaxis:

```
Program ::= Body
Body ::= VarDecList StatementList
VarDecList ::= (VarDec)*
VarDec ::= "var" Type VarList ";"
Type ::= id
VarList :: id (",” id)*
StatementList ::= Stm (";” Stm)* …
Stm ::=   id "=" CExp |
          "print" "(" CExp ")" |
          "if" CExp "then" Body ["else" Body] "endif" |
          "while" CExp "do" Body "endwhile"
```

Ademas contiene la implementación incompleta, pero compilable, del Typecheker `imp_typechecker.cpp`. Compilar y probar el programa con los ejemplos `ejemplo11.imp`, `ejemplo22.imp`, `ejemplo33.cpp` y, sobre todo, `ejemplo222cpp.`

En este laboratorio modificaremos el lenguaje IMP para poder incluir declaraciones de funciones. Esto implica que estaremos manejando **tipos para funciones**, además de los clásicos tipos para enetros y booleans. Para esto hemos decido representar tipos (antes representados por strings) por medio de una nueva clase ImpType definida por

La clase `ImpType` esta definida en `imp.hh` por

```
class ImpType {
public:
  enum TType { NOTYPE=0, VOID, INT, BOOL, FUN };
  TType ttype;
  vector<TType> types;
…
};
```

El campo `ttype` identifica al tipo como tipo básico (int, bool, void), un tipo de funciones, o un tipo invalido (default). En el caso de funciones, el vector `types` guarda los tipos de los parámetros y el tipo de retorno; estos tipos solo pueden ser básicos y `VOID` solo es permitido como tipo de retorno.

1.0 El TypeChecker: initializacion

El typechecker está definido por la clase `ImpTypeCheker` de la siguiente manera:

```
class ImpTypeChecker : public TypeVisitor {
public:
```

```
  ImpTypeChecker();
private:
  Environment<ImpType> env;
  ImpType booltype;
  ImpType inttype;
public:
  void typecheck(Program*);
  void visit(Program*);
….
  ImpType visit(BinaryExp* e);
…
};
```

Nótese que el environment `env` ahora guarda instancias de la clase `ImpType` y que el método `accept` para expresiones retorna `ImpType`.

Durante el proceso de typechecking haremos referencia con bastante frecuencia a los tipos int y bool. Es buena idea tenerlos a la mano en lugar de tener que crearlos cada que los necesitemos. Para esto, tenemos dos campos: `inttype` y `booltype`. Estos campos deben de inicializarse en el constructor de `ImpTypeChecker` de la siguiente manera:

ImpTypeChecker::ImpTypeChecker():inttype(), booltype() {

```
      inttype.set_basic_type("int");
      …. // bool?
}
```

Completar el constructor.

2.0 `ImpTypeChecker`.: Implementacion

El typecheker está implementado con un visitor. La implementación actual posee la funcionalidad para realizar la visita al AST (tree traversal) pero carece de los checks necesarios para completar el typechecking – se ha dejado comentado el código anterior como referencia.

Completar ImpTypeChecker.cpp con los type checks respectivos. Para esto será útil utilizar el método match() para comparar dos tipos:

```
ImpType type;
type.set_basic_type(vd->type);
if ( ! type.match(inttype) …)
```

3.0 Imp con funciones

Los nuevos programas de Imp tendrán, aparte de las definiciones de variables globales, una lista no vacia de declaraciones de funciones. Esto se ve reflejado por la nueva semántica:

```
Program ::= VarDecList FunDecList
FunDecList ::= (FunDec)+
FunDec ::= ”fun” Type id ParamDecList Body “endfun”
ParamDecList ::= “(“ epsilon | Type id (“,” Type id)* “)”
```

Además, se necesitara una nueva sentencia:

```
Stm ::= return ([CExp])
```

Y una nueva expresión:

Factor ::= … | id “(“ ExpList “)”
ExpList ::= épsilon | Exp

Para esto se necesitarán (o se necesitara modificar) las siguientes clases:

Program:
VarDecList var_decs; FunDecList fun_decs;
FunDecList:
list<FunDec*> fdlist;
FunDec:
string fname, rtype; list<string> types, vars; Body* body;
ReturnStm:
Exp* e;
FCallExp:
string fname, list<Exp*> args;

Actualizar el Parser y las clases en `imp.hh, imp.cpp` e `imp_printer.cpp` para implementar la nueva sintaxis e imprimir los nuevos programas. ¿Qué más necesitamos modificar?